REMARKS

This is a response to the non-final Office Action mailed on February 17, 2005.

Regarding paragraph 1 of the Office Action, Applicants respectfully disagree with the assertion that claims 13 and 14 are independent claims. A dependent claim contains a reference to a claim previously set forth and specifies a further limitation of the subject matter claimed. 35 U.S.C. §112. Claim 13 sets forth a computer program product comprising program code means for performing a method according to claim 1. Thus, the subject matter of the method of claim 1 is further limited by the requirement to provide a computer program product. Claim 14 is similarly a dependent claim, whose dependency has been corrected. Accordingly, these claims should not be treated as independent claims for calculating the extra claims surcharge. Reconsideration is respectfully requested.

Claims 2, 5 and 10 are amended to omit the word "preferably" to better comply with U.S. patent practice.

Claim 6 is amended in response to paragraph 2 of the Office Action.

Claim 14 is amended to depend on claim 13.

Claim 18 is amended to add a period at the end of the claim.

Referring to paragraph 3 of the Office Action, claims 1-19 have been rejected under 35 U.S.C. §102(a) as being anticipated by U.S. patent 6,230,316 to Nachenberg.

The present invention relates generally to providing information to an end user to allow the user to update first code to provide second code. This is achieved by providing generation instructions and a difference code to the user. The generation instructions are instructions that can be used by a first software archive generator to generate both the first and second code, such as at the location of a software provider. The difference code comprises the steps necessary to

arrive at the second code from the first code. The user employs a second software archive generator that, based on the generation instructions, uses the difference code and the first code to generate the second code.

Nachenberg provides incremental updating of an executable file that has been rebased or realigned. Nachenberg notes that, when an incremental software update is distributed, it may be rebased, which means memory addresses that appear in code and data segments are changed to accommodate the file being loaded into memory at a new base address. Furthermore, the software update may be realigned, which refers to moving the code and data segments within a file. See col. 1, line 66 to col. 2, line 10. Nachenberg states that such rebasing and realigning can result in unpredictable results when a software update occurs by binary patching. Binary patching is a technique for incrementally updating software in which the bits of the software that are different in a new version of the software are changed. See col. 1, lines 39-48 and lines 60-65.

Nachenberg avoids the unpredictable results that may occur due to rebasing or realigning by converting application files to a canonical form before performing an update. For example, as shown in Fig. 8, a version 100A of an application file is converted into a canonical version 100B. An update builder 122 uses a conventional binary patch file builder to determine a binary difference, in an update file 124C-B, between the version 100B file and the new version 100C file.

Update file 124C-B is then distributed to a user who installs it on computer 126, which also includes the canonical converter 120. A pre-update version of the file 100 on the computer 126 has been rebased and realigned, such that it is in state 100D. The canonical converter 120 on the computer 126 converts the file 100D to the canonical version 100B. The update file 124C-B

is then used by an updater 128 to produce the new version 100C of the file. The updater 128 can be any conventional binary patcher that applies the patch, e.g., update file 124C-B, to the canonical version 100B. See col. 5, lines 35-60.

In sum, Nachenberg provides a conventional binary patch file based on the difference between an updated file and a canonical version of a pre-update file. At the user's location, the binary patch file is applied to a canonical version of the pre-update file to derive the updated file.

Accordingly, Nachenberg fails to disclose or suggest various aspects of Applicants' invention. For example, Nachenberg fails to disclose or suggest providing to a software-acquiring entity, such as a user, a difference code that comprises the steps necessary to arrive from a first signed piece of code at a second signed piece of code. The binary patch update file 124C-B of Nachenberg (Fig. 8) does not indicate such steps, but only indicates a binary difference between two files.

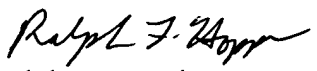
Nachenberg similarly fails to disclose or suggest the use of a second software archive generator that is fed with generation instructions that were used by a first software archive generator for generating both first and second pieces of code. Note that there is no mention in Fig. 8 of Nachenberg of the update file 100C being generated from the pre-update file 100A using generation instructions. Instead, it is only assumed that the update file 100C is somehow available, and the task Nachenberg is concerned with is providing a file that represents a binary difference between the file 100C and the canonical version 100B of file 100A.

Furthermore, Nachenberg makes no mention of signed code as set forth in Applicants' claims.

Accordingly, Applicants' claims are believed to be clearly patentable over Nachenberg.

In view of the above, each of the currently pending claims is believed to be in condition for allowance. The Examiner is respectfully requested to pass this application along to an early issue. The Examiner is invited to telephone the undersigned if there are any further issues to address.

Respectfully submitted,



Ralph F. Hoppin
Registration No. 38,494

SCULLY, SCOTT, MURPHY & PRESSER
400 Garden City Plaza, Suite 300
Garden City, New York 11530
(516) 742-4343

SF:RH